Automatic formal and formal verification

Yuri Friedman

Hani Dobrovitser
Agenda

**Formal verification:**
- What is formal verification (FV) ?
- FV vs. simulation
- Advantages and disadvantages
- Sweet spot
- Example
- Tolls
- Symbolic approach - ESP

**Symbolic approach - ESP**
What is formal verification - FV?

- Prove mathematically that a design obeys its specification
- Model checking method: Represent design as a finite state machine
- Automatically calculate truth or falsity of specification by traversing the state space
- Proving a property is showing that it holds for all possible input combinations, across all execution paths
What is formal verification - FV?
What is formal verification - FV?
What is formal verification - FV? (cont.)

- Input constraints capture assumptions on legal input stimulus
- Assertions are properties which capture design intent
- Model checking engine can prove or disprove whether a property is true for all possible states in the DUT
What is formal verification - FV? (cont.)

Input constraints → RTL + Assertions → Model Checking Engine

Fail – counterexample
Indeterminate
Proof – witness
## FV vs. Simulation

<table>
<thead>
<tr>
<th>FV</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>all legal input sequences</td>
<td>(large) set of particular cases</td>
</tr>
<tr>
<td>correctness expressed as set of general properties</td>
<td>correctness usually expressed per run (expected results)</td>
</tr>
</tbody>
</table>

### Diagram

- **Stimulus Generators**
- **Monitors & Checkers**
- Bar chart showing percentages for states S1, S2, S3, S4.
Advantages

- Improved verification quality
- Save time
- Less cost (less bugs, less tapeouts)
- No test required
- 100% coverage
- Reusable in simulation by assertions
Disadvantages

- Size limit
- Not always feasible
- Good for control checking but not for data
Sweet spot

- Small design
- Complex state space, many modes, concurrent events, control logic
- Arbiters, fairness mechanisms, detecting deadlocks
- Internal modules in complex block
Example of a module CPU interface

Assumptions
- req is a 1 clock cycle pulse
- a new req is not allowed until a previous req is acknowledged
- addr, wrNotRd, wrData are stable from req pulse until receipt of the ack pulse

Assertions
- ack is a 1 clock cycle pulse
- An ack can only occur after a req
- Each req can only have one ack
- ack must be asserted within 1 - 5 cycles of req
Example of a module CPU interface

```vhdl
assume property (     // no new req until ack
  @(posedge clk) req |=> !req[*0:$] ##0 ack);
assert property(    // ack within 1 to 5 cycles of req
  @(posedge clk) req |-> !ack[*1:4] ##1 ack);
```

![Timing diagram](image)
Tools

- RuleBase (IBM)
- JasperGold (Jasper)
- IFV (Cadence)
- Magellan (Synopsys)
Embedded Symbolic Processor (ESP)

- Formal techniques based on symbolic simulation
  - Provide fast functional coverage
  - Support all levels of design abstraction and styles

- Circuit Smarts technology
  - Enables the symbolic simulation to be applied at the transistor level

- Hierarchical Compression (HC) technology
  - Enables verification of 1B+ transistor-level designs
Symbolic Simulation

- Event-driven
  (like traditional logic simulation)
- Uses symbols (variables) as inputs instead of just 1s and 0s
- Propagates equations instead of discrete events
- Formal analysis, equivalence checking using generated equations
Symbolic Simulation

2\textsuperscript{nd} Clock
1 Symbolic Vector = $2^N$ Binary Vectors

4 Binary Vectors

- Z = 0 & 0
- Z = 0 & 1
- Z = 1 & 0
- Z = 1 & 1

4 Binary Vectors Replaced with 1 Symbolic Vector

1 Symbolic Vector

Z = s1 & s2
Coverage By Symbolic sim

N symbols = $2^N$ vectors

Vector Simulation

Symbolic Simulation
Useful or not ???

- Small design
- Small number of cycles
- Fast functional closer less vectors
- Not compatible with standard tools (coverage, ...)